Designing a Flexible Grid Enabled Scientific Modeling Interface

Mike Dvorak[1,2], John Taylor[1,2,3], and Sheri Mickelson[1,2]

[1] Mathematics and Computer Science Division,
Argonne National Laboratory, Argonne IL 60439
{dvorak, jtaylor, mickelso}@mcs.anl.gov
http://www-climate.mcs.anl.gov/
[2] Computation Institute,
University of Chicago, Chicago, IL, 60637
[3] Environmental Research Divisions
Argonne National Laboratory, Argonne IL 60439

Abstract. The Espresso Scientific Modeling Interface (Espresso) is a scientific model productivity tool developed for climate modelers. Espresso was designed to be an extensible interface to both scientific models and Grid resources. It also aims to be a contemporary piece of software that relies on Globus.org's Java CoG Kit for a Grid toolkit, Sun's Java 2 API and is configured using XML. This article covers the design and implementation of Espresso's Grid functionality and how it interacts with existing scientific models. We give specific examples of how we have designed Espresso to perform climate simulations using the PSU/NCAR MM5 atmospheric model. Plans to incorporate the CCSM and FOAM climate models are also discussed.

1 Introduction

The Espresso Scientific Modeling Interface (Espresso) is designed to utilize existing Grid computing technology to perform climate simulations [1]. Espresso is also a software tool that gives scientific model users the freedom to eliminate the mundane task of editing shell scripts and configuration files. It empowers the scientist to spend more time performing science and analyzing the output of climate simulations.

Espresso is tailored to the demands of the climate modeler. In the Mathematics and Computer Science (MCS) Division, we make global climate model runs using the Fast Ocean-Atmosphere Model (FOAM) [2]. We also create high resolution meteorological model runs for extended periods of time (e.g. hourly output for years over the United States at 10-52 km resolution) using a regional climate model. Making regional climate simulations requires a robust computing environment that is capable of dealing with resource faults inside complex model codes. Espresso is designed to meet the rigorous demands of a multi-architecture, terra-scale environment.

Moreover, Espresso strives to make the best use of contemporary technology by using: (1) the eXtensible Markup Language (XML) for system and graphical configuration; (2) the Globus.org Java CoG Kit for accessing Grid resources; (3) Sun's Java Web Start for software deployment; (4) a subset of the Apache Software Foundation's Jakarta Project utilities i.e. Regexp. Espresso is a pure Java program that is capable of being run from anywhere on the Internet.

Lastly, Espresso can be utilized by a wide variety of users, not just climate modelers. Espresso is designed for ease of use with different scientific models where model configuration is complex and Grid resources are required. Work is underway to incorporate both the FOAM and the Community Climate System Model [3] into Espresso.

This article focuses on the system side (non-graphical user interface (GUI)) and Grid application design of Espresso. For a detailed look at the design of the GUI, see [4]. It should also be mentioned that Espresso is a second generation tool with the Argonne Regional Climate Workbench [5] as its predecessor. Espresso is different from the Climate Workbench in that: (1) its client side is pure Java; (2) it can be an interface for multiple models; (3) it is fully configurable via text (XML) files; (4) it can run anywhere on the Internet. Lessons learned from the design of the Climate Workbench and its applications have contributed significantly to the design of Espresso.

2 An Example Scenario Using Espresso

While the details of Grid computing for climate science are detailed in papers like [6], the following scenario provides insight on how a Grid enabled interface can be utilized to help climate scientists perform climate simulations. Figure 1 provides an example situation of how a climate scientist might interact with Espresso.

To the user, the most important part of Espresso is the GUI. Espresso's GUI hides all of the implementation details behind buttons and tell-tale output indicators. Users are not forced to learn all of the climate model's idiosyncrasies. Instead, they can concern themselves with the relevant model parameters. The GUI is a secure proxy to the Grid resources. From Espresso, the user can command the supercomputing, storage and analysis resources the user has access to with their authenticated certificate. The user only needs to authenticate once to obtain access to all Grid resources which they are authorized to use. This is the Internet equivalent to logging on to a single computing system.

After the user authenticates, they enter their climate simulation parameters via Espresso's GUI. Figure 1 shows the a scenario in which the climatologist wants to access a historical archive of climate data on a remote server (which could contain terabytes of data) to obtain initial and boundary conditions for a regional climate simulation. Inside the GUI the user simply specifies the historical data set via a combo box. The run dates, geographical grid, and model parameters to run for the simulation is set in text boxes. The user then submits

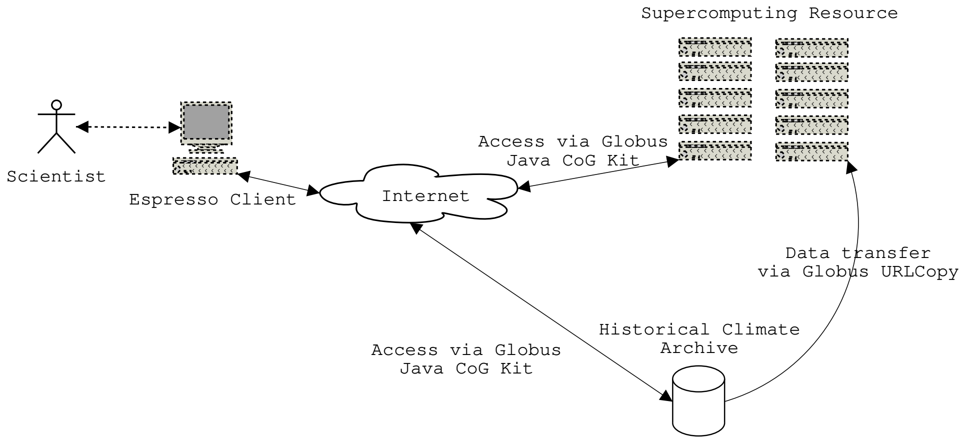

Fig. 1. An Example Scenario Using Espresso to Perform a Climate Run

the job to the Espresso server side component (which could be located anywhere on the Internet) for execution.

Eventually, the Espresso server side component will know how to use a Grid toolkit to obtain the climate data from a remote historical archive. The supercomputing resource will then use third party data transfer to obtain the climate data and proceed with the climate simulation. After the simulation is finished, the Grid toolkit could also be used to analyze and move the data to a different storage server via the Grid. For an overview of Grid computing infrastructure, see [1].

3 Modeling Interface Software Design

Table 1 highlights the most significant design requirements that were required for regional and global climate modeling. These requirements also help to make Espresso sufficiently flexible so that it could be used by other scientific modeling systems.

3.1 Special Considerations for Climate Modeling

Climate modeling places unbounded demands on supercomputing resources. Regional climate modeling intensifies demand on computing resources by increasing spatial resolution and the frequency of atmospheric/oceanographic data output. In MCS, initial and boundary condition files typically run on the gigabyte scale with raw data output running on the terabyte scale. Climate modeling systems also consist of several different data preprocessing programs and executables with many different build options. An effective interface for a climate modeling

Design Requirement	Implementation Solution
Grid enabled	Globus.org Java CoG Kit
System-side easily configurable	XML
GUI easily configurable	XML/Java Swing
Distributable case studies	XML
Easy package deployment	Java Web Start
Run anywhere on Internet	Globus Java CoG Kit/ Java Web Start

Table 1. General design requirements and implementation solutions in Espresso

system must be able to work with all of these preprocessing and build programs in a dynamic manner.

The Penn State University/National Center for Atmospheric Research Mesoscale Model 5 (MM5) is a good example of a scientific model that pushes a computing resource to its limits. We use MM5 for high resolution, regional climate runs. Most of the challenges of running MM5 evolved from porting the atmospheric model to different architectures i.e. MCS's Chiba City, a 512-processor Linux cluster. Other high performance problems are derived from running MM5's complex system of data preprocessing programs. The model is also written in Fortran 77 so variables are not dynamically allocated. Having to run a scientific modeling system with a complex build and preprocessor system placed a high quality design requirement on Espresso. A good review of MM5 is given in [7] [8] [9] and [10].

3.2 Making Espresso Work With Multiple Models

In order to make Espresso usable for a wide group of scientific models, it was necessary to make three broad assumptions:

- Large scientific modeling systems are configured by editing text configuration files.
- By editing these text files and replacing them within the system, the original scientific model can be used in the way that its designers intended.
- No code modifications to the original software.

By stating these underlying assumptions during the design and implementation of Espresso, it has been easy to determine which models will be able to use this interface. An important side effect of these assumptions is that Espresso can easily be used with a scientific model not originally designed to have a GUI. This is the situation for many older Fortran scientific codes.

3.3 General Espresso System Requirements

In order to accommodate the needs of a wide variety of users, Espresso must be extensible. We needed both a Grid functionality component and a GUI that was easily configurable using an input text file. The original Climate Workbench was limited to running only the MM5 model and contained hard-coded fields specific to MM5. It would have been very difficult to extend this interface to new models. Therefore, the interface needed to be customizable via a ASCII text file. Ideally this text configuration file would be written in XML to take advantage of freely available parsing tools.

We required the use of object oriented design and programming techniques in order to incorporate extensibility. The original Climate Workbench modeling interface was written such that it was nearly impossible to extend the functionality of the interface. Along with this design paradigm came the desire to reuse as much code as possible through the inheritance of key scientific modeling components i.e. model variable data structures and general text processing algorithms. Model specific tasks such as the regular expression syntax would be sub-classed.

The most critical design feature to building a "wrapper modeling interface" was embracing the scientific model without structural changes. This approach has substantial advantages with regard to code maintenance. Figure 2 illustrates how Espresso accomplishes this task. Contained within Espresso are only the model's original configuration files that will be used as a template. In step 2, Espresso has modified the configuration files with error checking. Step 3 places the files on the server side supercomputing resource using the Globus URL Copy functionality. These configuration files are "pushed" into place like the final pieces of puzzle, allowing the model to be run as originally intended.

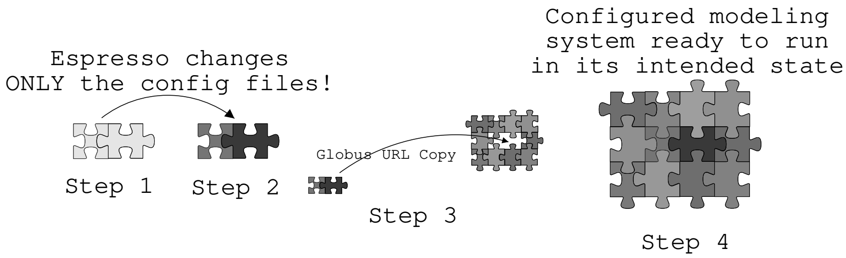

Fig. 2. Espresso runs the remote system by editing the configuration files. These configuration files are then moved back into their original location. The scientific model can be executed in the way intended by the original designers. No structural changes are made to the scientific modeling code when using Espresso.

By making no changes to the scientific modeling system, upgrades can also be performed with minimal effort. This design requirement limited us to only having the configurable text files on the remote system and then copying these files to the server, in the appropriate location. Updating versions of the model code can be achieved with minimal effort using this approach.

The Climate Workbench could be run only on specific machines in MCS. The old version assumed that the Network File System (NFS) was available.

Unfortunately, this limited the interface to run only within the MCS network. We wanted to be able to run Espresso from anywhere via a Java enabled browser. Espresso would have to run as a stand alone application and access all resources via its Grid interface. The Globus Java CoG kit made all of this functionality possible.

Some users may desire to run a non-GUI version of the modeling interface, i.e. a text only version. For testing purposes, this was also a very important feature. Other users may want to perform special Grid computing tasks that would not be feasible within a GUI. This would allow experienced users to take advantage of all of Java and Globus tools described above without the need to enter data via the Interface. Error checking occurs in the GUI so this feature would be lost in the non-GUI version ([4] discusses Espresso's GUI error checking in detail).

3.4 Espresso Server Side Component

The implementation of Espresso's server side component uses several shell scripting languages (TCSH, Python, BASH). For testing and modular purposes, we needed all of these shell scripts to run independently of Espresso. We also wanted to rid the server side of scripts that took long command line arguments. Consequently, we developed additional helper scripts that discovered information about the variables needed to run other scripts e.g. the start and end dates of the model simulation.

4 Espresso Implementation

4.1 Grid Utilization via the Globus Java CoG Kit

In order to make the Espresso client pure Java code, we needed a Grid toolkit that was implemented in Java. The Globus.org group had available a Java implementation of its Commodity Grid (CoG) Kit [11] [12]. The availability of the Java CoG Kit allowed us to integrate all of the Grid functionality that we needed with its Java API. The Java CoG Kit has all the necessary packages in a single Java ARchive (JAR) which can be distributed with the modeling interface. The Java CoG Kit communicates with the C implementation of the Globus Toolkit to perform tasks on the server side.

The two components utilized in the Globus Toolkit are the Globus Resource Allocation Manager (GRAM) and the GridFTP components. GRAM is used to run jobs remotely on different computation resources [13]. In order to execute a job on a resource machine, users use the Resource Specification Language (RSL) to tell the GRAM server what type of job is to be run. Typically, the RSL string sent to a supercomputing resource includes the name of the executable and the location of the standard out and error files (which can be located remotely).

Editing all of the model configuration files on the system side required that we transfer these files to the server side. We used the CoG Kit's Globus URLCopy from the GridFTP [14] component to provide this functionality. The URLCopy class allows both second and third party file transfers in a fashion similar to that of the common FTP client. Authentication on the remote server(s) is handled with the Globus security credentials and there is no need to type in another password once the original Grid Proxy has been established. We plan to use URL Copy's 3rd party copy functionality in future versions of Espresso to move large files from server to server (to which we may add additional error checking).

4.2 Creating a Java Based SED

The Unix "Stream EDitor" (SED) is a commonly used tool to edit streams of text. Since we assumed the scientific models were configured by editing text files, we needed the equivalent functionality of SED operations in Java. The first version of the interface used SED on the Unix server side to edit files.

The Java Foundation Class (JFC) provides the java.util.StringTokenizer class that allows one to parse tokens. Significant additional coding and testing had to be undertaken to mimic SED's functionality. The Apache Software Foundation's Jakarta Project provides us with a regular expressions package written in Java (appropriately named "Regexp") to use with the Java IO package. This allows us to build regular expressions for each model variable. MM5 for example, uses the "Regexp" regular expression "^[.*\s—](" + v.getVariableName() + ")\s{1,}=\s{1,}(\d{0,})" to search through the variables in a Fortran name list.

5 Delivering Espresso to the User Community

5.1 Obtaining Access to Grid Resources

In order to use Espresso with Globus Grid resources, you need to establish your Globus credentials. You must first obtain these Globus credentials from the Certificate Authority [16]. Next, you need to have your name added to the "grid-map" on a Globus resource e.g. a mass storage device or a high performance computer. The number of resources that you have access to is limited only by your need to obtain permission to access the resource and Globus being installed on the system.

To use these Grid resources, you are required to validate your Globus credentials via the "grid-proxy-init" utility which asks for you password and validates your certificate for a fixed amount of time (12 hours by default). Once the "grid-proxy-init" is performed, you have complete access to all of your computational resources through the Globus toolkit. Espresso utilizes this functionality by copying files to a file server and running the scientific model on a different machine.

5.2 Accessing Espresso Technology

Delivering updated software to our user community was a concern from the start of the project. We wanted to distribute Espresso with minimal difficulty for both

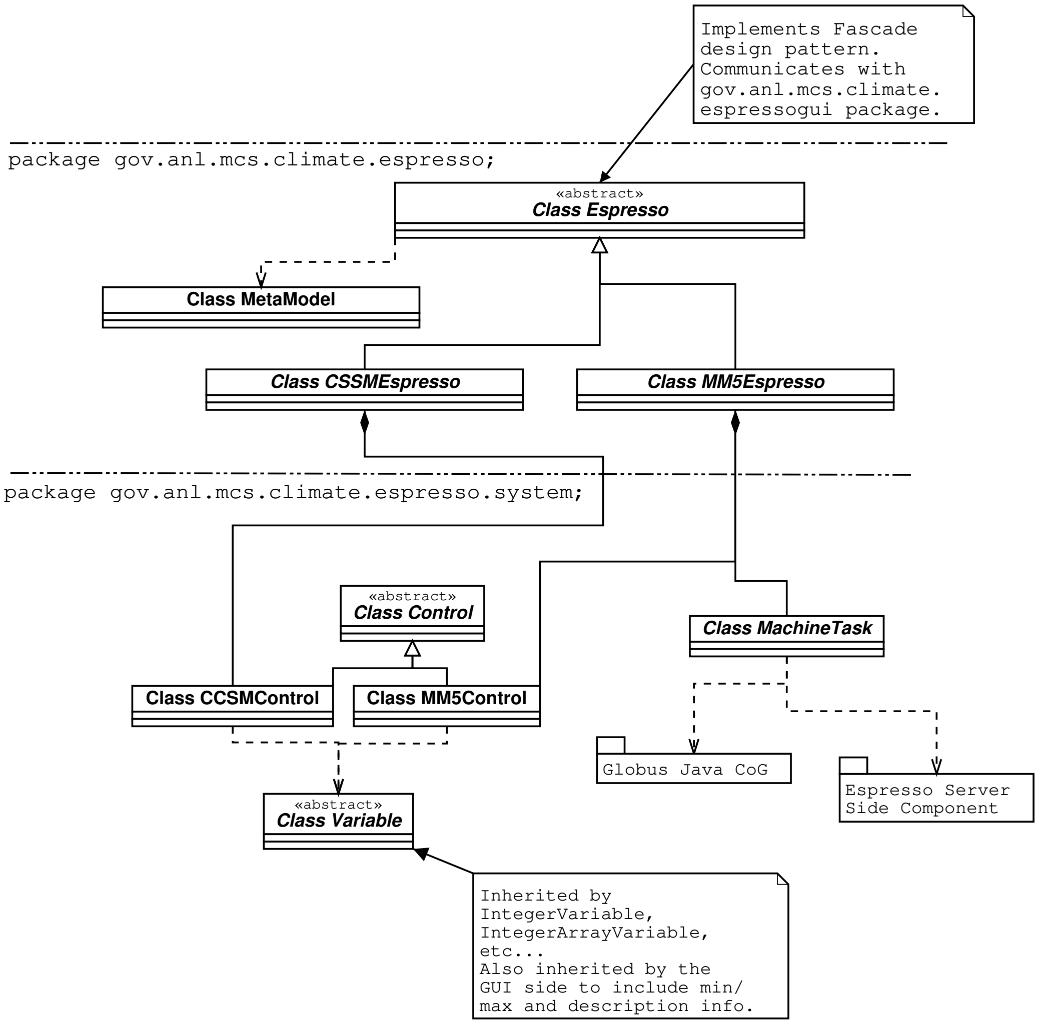

Fig. 3. UML Description of the Espresso System Side Design (Facade design pattern [15])

us (the developers) and the users. Fortunately Sun Microsystems, Inc. has a web-based software solution called Java Web Start [17]. Once users install Web Start (available for Windows, Solaris, Linux and Macintosh OS X), programs are installed, updated and executed via an Internet or desktop link. If Espresso is updated on the web server, Web Start detects a newer version and downloads the update.

Once a historical climate data archive is set up (similar to the diagram in Figure 1), we plan on creating a "case study repository" of XML files. Having a repository of XML files would allow other users to replicate or alter the parameters of other scientists model simulations. Other users could create case studies of interesting weather/climate events and exchange the XML files with other model users. All users could potentially have access to the historical climate data referenced in the "case studies" via the Grid.

6 Conclusion and Future Work

The initial version of the Espresso Scientific Modeling Interface, a scientific modeling productivity tool incorporating Grid technology has been developed. Creating a multipurpose scientific modeling interface that could be applied to many different scientific models was a challenging task. Using existing tools such as the Apache Software Foundation's XML parsers and regular expressions packages, Globus.org's Java CoG Kit and Sun's Web Start technology has allowed us to produce a high quality scientific model interface.

Espresso was intended to be developed iteratively, with the initial focus on the MM5 atmospheric model. Current efforts are being directed toward incorporating FOAM and the CCSM climate models. This will help us to abstract common components of scientific model simulation and analysis. In the coming year, we plan on updating the design of Espresso to simplify the task of adding scientific models to the interface. With increased Grid functionality, Espresso could become an important software tool for performing Grid based climate simulations.

Acknowledgments

We thank that the staff the Mathematics and Computer Science Division and the Computation Institute. We would also like to thank Gregor von Laszewski and Jarek Gawor of the Globus.org group for helping us use the Globus Java CoG Kit. This work was supported in part by the Office of Biological and Environmental Research, U.S. Department of Energy, under Contract W-31-109-ENG-38, and by the NSF Information Technology Research Grant, ATM-0121028.

References

1. Ian Foster and Carl Kesselman. *The Grid: Blueprint For A New Computing Infrastructure.* Morgan Kaufmann Publishers, 1999.

2. Robert Jacob, Chad Schafer, Ian Foster, Michael Tobis, and John Anderson. Computational Design and Performance of the Fast Ocean Atmosphere Model, Version One. In *Computational Science - ICCS 2001*, volume Part I, pages 175–184. International Conference on Computational Science, Springer, May 2001.
3. The CCSM Home Page. `http://www.ccsm.ucar.edu/`.
4. Sheri Mickelson, John Taylor, and Mike Dvorak. Simplfying the Task of Generating Climate Simulations and Visualizations. Submitted to the *2002 International Conference on Computational Science*.
5. John Taylor. Argonne Regional Climate Workbench. `http://www-climate.mcs.anl.gov/proj/climate/public_html/`.
6. John Taylor, Mike Dvorak, and Sheri Mickelson. Developing GRID based infrastructure for climate modeling. Submitted to the *2002 International Conference on Computational Science*.
7. F. Chen and J. Dudhia. Coupling an Advanced Land-Surface/Hydrology Model with the Penn State/NCAR MM5 Modeling System: Part I: Model Implementation and Sensitivity. *Monthly Weather Review*, 2001. "See also Pennsylvania State University/National Center for Atmospheric Research, MM5 Home Page" `http://www.mmm.ucar.edu/mm5/mm5-home.html`.
8. F. Chen, K. Mitchell, J. Schaake, Y. Xue, H. L. Pan, V. Koren, Q. Y. Duan, K. Elk, and A. Betts. Modeling Land-Surface Evaporation by Four Schemes and Comparison with FIFE Observations. *Journal of Geophysical Research*, 101:7251–7268, 1996.
9. J. Dudhia. A Nonhydrostatic Version MM5 of the Penn State/NCAR Mesoscale Model: Validation Test and Simulation of an Atlantic Cyclone and Cold Front. *Monthly Weather Review*, 121:1493–1513, 1993.
10. G. A. Grell, J. Dudhia, and D. R. Stauffer. The Penn State/NCAR Mesoscale Model (MM5). Technical Report NCAR/TN-398+STR, National Center for Atmospheric Research, 1994.
11. Gregor von Laszewski, Ian Foster, Jarek Gawor, and Peter Lane. A Java Commodity Grid Kit. *Concurrency and Computation: Practice and Experience*, 13:645–662, 2001.
12. The Globus Toolkit CoG Kit Homepage. `http://www.globus.org/cog/java/`.
13. Karl Czajkowski, Ian Foster, Nicholas Karonis, Carl Kesselman, Stuart Martin, Warren Smith, and Steve Tueke. A Resource Management Architecture for Metacomputing Systems. Technical report, Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, 1998. pp. 62-82.
14. GridFTP:Universal Data Transfer for the Grid. Technical report, Globus Project, September 2000.
15. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.
16. The Globus Homepage. `http://www.globus.org//`.
17. Java Web Start Home Page. `http://java.sun.com/products/javawebstart/`.